23장. DBMS와 RecyclerView

23.1. SQLite을 이용한 영속화

- SQLite (www.sqlite.org)는 오픈소스로 만들어진 관계형 데이터베이스
- data/data/[package_name]/databases 에 저장

23.1.1. SQLiteDatabase 클래스

- SQL 문 수행 은 이 클래스의 함수를 이용

```
val db = openOrCreateDatabase("memodb", Context.MODE_PRIVATE, null)
```

23.1. SQLite을 이용한 영속화

- execSQL(sql: String): insert, update 등 select 문이 아닌 나머지 SQL 수행
- rawQuery(sql: String, selectionArgs: Array<String>): select SQL 수행

```
db.execSQL("insert into tb_memo (title, content) values (?,?)", arrayOf<String>("hello", "world"))
```

```
val cursor = db.rawQuery("select title, content from tb_memo order by _id desc limit 1", null)
```

- Cursor는 선택된 행(row)의 집합 객체

- moveToNext(): 순서상으로 다음 행 선택
- moveToFirst(): 가장 첫 번째 행 선택
- moveToLast(): 가장 마지막 행 선택
- moveToPrevious(): 순서상으로 이전 행 선택

```
while (cursor.moveToNext()){
    titleView.setText(cursor.getString(0));
    contentview.setText(cursor.getString(1));
}
```

23.1. SQLite을 이용한 영속화

23.1.2. SQLiteOpenHelper 클래스

- 테이블 생성이나 스키마 변경 등의 작업

```
class DBHelper(context: Context): SQLiteOpenHelper(context, "memodb", null, 1) {

    override fun onCreate(db: SQLiteDatabase) {
        //...
    }
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        //...
    }
}
```

- onCreate( ): 앱이 설치된 후 SQLiteOpenHelper가 최초로 이용되는 순간 한 번 호출
- onUpgrade( ): 데이터베이스 버전이 변경될 때마다 호출

```
val helper = DBHelper(this)
val db = helper.writableDatabase
```

23.1. SQLite을 이용한 영속화

23.1.3. insert( ), query( ), update( ), delete( ) 함수 이용

- insert(table: String, nullColumnHack: String, values: ContentValues)
- update(table: String, values: ContentValues, whereClause: String, whereArgs: Array<String>)
- delete(table: String, whereClause: String, whereArgs: Array<String>)
- query(table: String, columns: Array<String>, selection: String, selectionArgs: Array<String>, groupBy: String, having: String, orderBy: String, limit: String)

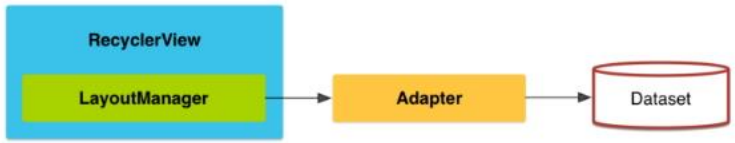
```
val values = ContentValues()
values.put("name", "kkang")
values.put("phone", "0100000")
db.insert("USER_TB", null, values)
```

```
val c = db.query("USER_TB", arrayOf("name", "phone"),
        "ID=?", arrayOf("kkang"), null, null, null)
```

23.2. RecyclerView

23.2.1. RecyclerView 소개

- support:recyclerView-v7 라이브러리로 제공
    - implementation **'com.android.support:recyclerview-v7:26.1.0'**



- Adapter: RecyclerView 항목 구성
- ViewHolder: 각 항목 구성 뷰의 재활용을 목적으로V iew Holder 역할
- LayoutManager : 항목의 배치
- ItemDecoration: 항목 꾸미기
- ItemAnimation: 아이템이 추가, 제거, 정렬될 때의 애니메이션 처리

23.2. RecyclerView

23.2.2. Adapter, ViewHolder

- ViewHolder의 역할은 항목을 구성하기 위한 뷰들을 findViewById 해주는 역할

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

```
private class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val itemTextView=itemView.itemTextView
}
```

23.2. RecyclerView

```
private class MyAdapter(private val list: List<String>) : RecyclerView.Adapter<MyViewHolder>() {

    override fun onCreateViewHolder(viewGroup: ViewGroup, i: Int): MyViewHolder {
        val view = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_main, viewGroup, false)
        return MyViewHolder(view)
    }

    override fun onBindViewHolder(viewHolder: MyViewHolder, position: Int) {
        val text = list[position]
        viewHolder.itemTextView.text= text
    }

    override fun getItemCount(): Int {
        return list.size
    }
}
```

```
recyclerView.layoutManager = LinearLayoutManager(this);
recyclerView.adapter = MyAdapter(list);
```

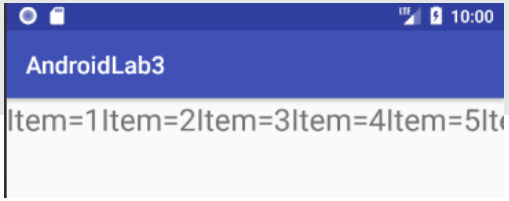
23.2. RecyclerView

23.2.3. LayoutManager

- 항목을 어떻게 배치할 것인가를 결정

- LinearLayoutManager: 수평, 수직으로 배치
- GridLayoutManager: 그리드 화면으로 배치
- StaggeredGridLayoutManager: 높이가 불규칙한 그리드 화면으로 배치

```
val linearManager = LinearLayoutManager(this)
linearManager.orientation = LinearLayoutManager.HORIZONTAL
recyclerView.layoutManager = linearManager
```

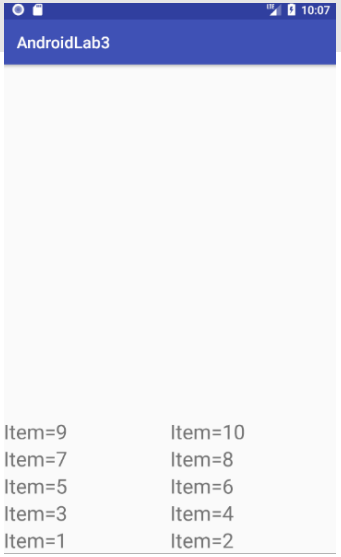



```
val gridManager = GridLayoutManager(this, 2)
recyclerView.layoutManager = gridManager
```

23.2. RecyclerView

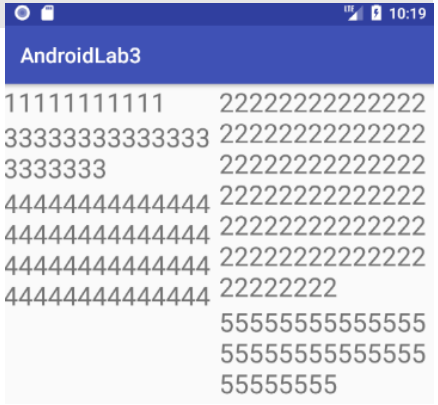
```
val gridManager = GridLayoutManager(this, 2, GridLayoutManager.HORIZONTAL, false)
recyclerView.layoutManager = gridManager
```


```
val gridManager = GridLayoutManager(this, 2, GridLayoutManager.VERTICAL, true)
recyclerView.layoutManager = gridManager
```



23.2. RecyclerView

```
val sgManager = StaggeredGridLayoutManager(2, StaggeredGridLayoutManager.VERTICAL)
recyclerView.layoutManager = sgManager
```



23.2. RecyclerView

23.2.4. ItemDecoration

- 항목을 다양하게 꾸미기 위해 사용

- onDraw: 항목을 배치하기 전에 호출
- onDrawOver: 모든 항목이 배치된 후에 호출
- getItemOffsets: 각 항목을 배치할 때 호출

```
class MyItemDecoration : RecyclerView.ItemDecoration() {
    override fun getItemOffsets(outRect: Rect, view: View, parent: RecyclerView,
                                state: RecyclerView.State) {
        super.getItemOffsets(outRect, view, parent, state)
        //항목의 index 값 획득
     val index = parent.getChildAdapterPosition(view) + 1
        if (index % 3 == 0)
        //left, top, right, bottom
            outRect.set(20, 20, 20, 60)
        else
            outRect.set(20, 20, 20, 20)
        view.setBackgroundColor(0xFFECE9E9.toInt());
        ViewCompat.setElevation(view, 20.0f)
    }
}
```


```
recyclerView.addItemDecoration(MyItemDecoration())
```